SUMMARY OF ANSI X9.44
Public Key Cryptography for the Financial Services Industry: Key Establishment Using Factoring Based Public Key Cryptography

1. Scope

ANSI X9.44 defines key establishment schemes that employ asymmetric cryptographic techniques. The arithmetic operations involved in the operation of the schemes take place in the algebraic structure of a multiplicative group of a finite ring. Both key agreement and key transport schemes are specified, but only in an informative appendix in the current draft. The schemes may be used by two parties to compute shared keying data that may then be used by symmetric schemes to provide cryptographic services, e.g. data confidentiality and data integrity.

2. Contents

The document defining ANSI X9.44 is still in development and some of its content has yet to be decided. Currently, the body of the document provides the underlying mechanisms for constructing the public keys (although it cross-references ANSI X9.80[1] and ANSI X9.82[2] for many algorithms – both are currently under development), the mechanisms for handling bit strings, the computational mechanisms for encryption and decryption, and the mechanisms for formatting messages. It does not, however, discuss the actual protocols for key agreement and key transport. Instead, the appendix provides ***some*** protocols for doing so.

A major issue, yet to be decided is whether extended protocols for key transport and key agreement should be in this (draft) standard (such as protocols to provide forward secrecy or entity authentication) or whether those should be relegated to ANSI X9.70[3]. Note that many of the protocols in ANSI X9.42[4] and ANSI X9.63[5] do not apply here, as the schemes and primitives for discrete logarithm cryptography have substantially different properties than those for cryptography based on integer factorization.

Another issue to be decided is whether to make this standard a general standard for encryption using factoring based cryptography rather than specifically for key transport and agreement. Of course, mechanisms would also be provided for the latter, based on the former.

The asymmetric key establishment schemes in ANSI X9.44 are used by an entity *U* who wishes to establish a symmetric key with another entity *V*. Each entity has its own separate public key pair. If *U* and *V* simultaneously execute a scheme with either one party using the other party's public key to transport an asymmetric key to the other party or with both parties using each others' public key to jointly agree on an asymmetric key, then at the end of the execution of the scheme, *U* and *V* will share keying data. The keying data can then be used to supply keys for symmetric algorithms.

[1] Prime Number generation, Primality Testing and Primality Certificates
[2] Random Number Generation
[3] Management of Symmetric Keys Using Public Key Algorithms
[4] Agreement of Symmetric Keys Using Discrete Logarithm Cryptography
[5] Key Agreement and Key Transport Using Elliptic Curve Cryptography

The standard contains specifications for:

- Key pair generation and key validation
- Cryptographic hash (although it references the forthcoming SHA-2)
- Bit string to integer conversion (and vice-versa)
- RSA encryption and decryption primitives
- Rabin-Williams encryption and decryption primitives
- OAEP (Optimal Asymmetric Encryption Padding), the method for padding a message before transport or agreement.
- The ES-OAEP encryption scheme, including encryption and decryption operations
- Mask Generation Functions used in OAEP
- Example mechanisms for how to use the encryption scheme to achieve key transport or agreement
- Key construction functions used in key agreement

In addition, for each scheme specified in the standard, an assessment of the security that is afforded by the scheme is provided.

3. Cryptographic Ingredients

3.1 Factoring Based Keying Material

The keying material needed includes the following elements.

- p,q two large primes
- n their product (n is public)
- e a public exponent
- d a private key. Note that d and e are related. How they are related depends on whether e is odd or even.

The public key parameters n and e may be distributed, for example, in a public key certificate.

3.2 Key Pair Generation and Public Key Validation

ANSI X9.44 provides primitives for public key generation. The primitives differ depending on the characteristics of the public exponent, e.

Each entity shall secretly and randomly select two distinct positive primes, p and q, such that the following conditions are true:

Constraints on p and q relative to e are:

— If e is odd, then e shall be relatively prime to both p-1 and q-1.

— If e is even, then p shall be congruent to 3 mod 8, q shall be congruent to 7 mod 8, and e shall be relatively prime to both $\frac{(p-1)}{2}$ and $\frac{(q-1)}{2}$.

The public encryption exponent e is selected prior to generating the private prime factors, p and q.

The private decryption exponent, d, shall be a positive integer value such that $d > 2^{512+64s}$, where s is an integer $s \geq 0$ [[s is the same integer as for the modulus]], (i.e. the length of the private decryption exponent must be at least half of the length of the modulus n). d is calculated as follows:
— If e is odd, then $d = e^{-1}$ mod (LCM (p-1, q-1)).
— If e even, then $d = e^{-1}$ mod (½ LCM (p-1, q-1)).

The method of generating p and q is relegated to ANSI X9.80.

3.3 IFEP1 Primitive: RSA-based encryption

IFEP1 ((n, e), m)

Input:	(n, e)	public key with e odd
	m	message representative, an integer between 0 and n-1
Output:	c	ciphertext representative, an integer between 0 and n-1; or "message representative out of range"

Assumptions: public key (n, e) is valid

Steps:

1. If the message representative m is not between 0 and n-1, output "message representative out of range" and stop.
2. Let $c = m^e$ mod n.
3. Output c.

3.4 IFEP2 Primitive: Rabin-Williams encryption

IFEP2 ((n, e), m)

Input:	(n, e)	public key with e even
	m	message representative, a nonnegative integer that is at most (n–5)/4.
Output:	c	ciphertext representative, an integer between 0 and n-1; or "message representative out of range"

Assumptions: public key (n, e) is valid

Steps:
1. If m is not between 0 and (n-5)/4, then output “message representative out of range” and stop.
2. Let $s = 4m + 4$.
3. If $\left(\frac{s}{n}\right) = 1$, then let $c = s^e \mod n$.
4. Else, if $\left(\frac{s}{n}\right) = -1$, then let $c = (s/2)^e \mod n$.

Output c.

3.5 IFDP1 – RSA based decryption

IFDP1 ((n, d), c)

Input:	(n, d)	private key
	c	ciphertext representative, an integer between 0 and n-1
Output:	m	message representative, an integer between 0 and n-1; or “ciphertext representative out of range”

Assumptions: private key (n, d) is valid

Steps:
1. If the ciphertext representative c is not between 0 and n-1, output “ciphertext representative out of range” and stop.
2. Let $m = c^d \bmod n$.
3. Output m.

3.6 IFDP2 – Rabin-Williams decryption

IFDP2 ((n, d), c)

Input:	(n, d)	private key
	c	ciphertext representative, an integer between 0 and n-1
Output:	m	message representative, an integer between 0 and n-1; or “ciphertext representative out of range”

Assumptions: private key (n, d) is valid

1. If the ciphertext representative, c, is not between 0 and n-1, then output "ciphertext representative out of range" and stop.

2. Let $t = c^d \mod n$.

3. Recovery of the padded message, s:

a) If $t \equiv 0 \mod 4$, then let $s = t$.

b) Else, if $t \equiv 1 \mod 4$, then let $s = n-t$.

c) Else, if $t \equiv 2 \mod 4$, then let $s = 2t$.

d) Else, if $t \equiv 3 \mod 4$, then let $s = 2(n-t)$.

4. If $s \neq 4 \mod 256$ output "error" and stop.

5. Let $m = (s-4)/256$.

6. Output m.

3.7 OAEP Encryption/Decryption Scheme

ENCRYPTION

ES-OAEP-ENCRYPT ((n, e), M, P)

Input:	(n, e)	recipient's public key
	M	message to be encrypted, a bit string of length at most $k - r - 1 - 2hLen$, where k is the length in bits of the modulus n, $hLen$ is the length in bits of the hash function output for EME-OAEP and $r = 8$ if e is odd and $r = 16$ if e is even.
	P	encoding parameters, a bit string that may be empty
Output:	C	ciphertext, a bit string of length k; or "message too long"
Assumptions:	public key (n, e) is valid	

Steps:

1. Apply the EME-OAEP encoding operation to the message M and the encoding parameters P to produce an encoded message EM of length $k - r$ bits, where r is as defined above EM = EME-OAEP-ENCODE (M, P, $k - r$)

   If the encoding operation outputs "message too long," then output "message too long" and stop.

2. Convert the encoded message EM to an integer message representative m:

$$m = \text{BS2IP}(EM)$$

3. If e is odd then apply the IFEP1 encryption primitive to the public key (n, e) and the message representative m to produce an integer ciphertext representative c:

$$c = \text{IFEP1}\ ((n, e), m)$$

   Else (if e is even), apply the IFEP2 encryption primitive to the public key (n, e) and the message representative m to produce an integer ciphertext representative c:

$$c = \text{IFEP2}\ ((n, e), m)$$

4. Convert the ciphertext representative c to a ciphertext C of length k bits:

$$C = \text{I2BSP}\ (c, k)$$

5. Output the ciphertext C.

This section describes the encoding method EME-OAEP. [[May be clearer if in a separate section]]

EME-OAEP-ENCODE (*M*, *P*, *emLen*)

Options:	*Hash*	hash function (*hLen* denotes the length in bits of the hash function output)
	MGF	mask generation function
Input:	*M*	message to be encoded, a bit string of length at most *emLen*-2*hLen-1*
	P	encoding parameters, a bit string
	emLen	intended length in bits of the encoded message, at least 2*hLen*+1
Output:	*EM*	encoded message, a bit string of length *emLen*; or “message too long” or “parameter string too long”

Steps:

1. If the length of P is greater than the input limitation for the hash function (2^{64}-1 bits for SHA-1) then output “parameter string too long” and stop.

2. If $||M|| > emLen\text{-}2hLen\text{-}1$ then output “message too long” and stop.

3. Generate a bit string PS consisting of $emLen\text{-}||M||\text{-}2hLen\text{-}1$ zero bits. The length of PS may be 0.

4. Let $pHash = Hash(P)$, a bit string of length $hLen$.

5. Concatenate $pHash$, PS, the message M, and other padding to form a data block DB as

$$DB = pHash \mid\mid PS \mid\mid 1 \mid\mid M$$

6. Generate a random bit string $seed$ of length $hLen$.

7. Let $dbMask = MGF(seed, emLen – hLen)$.

8. Let *maskedDB* = *DB* ⊕ *dbMask*.

9. Let *seedMask* = *MGF*(*maskedDB*, *hLen*).

10. Let *maskedSeed* = *seed* ⊕ *seedMask*.

11. Let *EM* = *maskedSeed* || *maskedDB.*

12. Output *EM*.

DECRYPTION:

ES-OAEP-DECRYPT (*K*, *C*, *P*)

Input:	(*n*, *d*)	recipient's private key
	C	ciphertext to be decrypted, a bit string of length *k*, where *k* is the length in bits of the modulus *n*
	P	encoding parameters, a bit string that may be empty
Output:	*M*	message, a bit string of length at most $k - r - 1 - 2hLen$, where *hLen* is the length in bits of the hash function output for EME-OAEP and $r = 8$ if *e* is odd and $r = 16$ if *e* is even; or "decryption error"

Steps:

1. If the length of the ciphertext *C* is not *k* bits, output "decryption error" and stop.

2. Convert the ciphertext *C* to an integer ciphertext representative *c*:

$$c = \text{BS2IP}(C)$$

3. If (*n*, *d*) is a private key that corresponds to a public key with odd *e*, apply the IFDP1 decryption primitive to the private key (*n*, *d*) and the ciphertext representative *c* to produce an integer message representative *m*:

$$m = \text{IFDP1}((n, d), c)$$

If IFDP1 outputs "ciphertext representative out of range," then output "decryption error" and stop.

If (*n*, *d*) is a private key that corresponds to a public key with even *e*, apply the IFDP2 decryption primitive to the private key (*n*, *d*) and the ciphertext representative *c* to produce an integer message representative *m*:

$$m = \text{IFDP2}((n, d), c)$$

If IFDP2 outputs "ciphertext representative out of range," then output "decryption error" and stop.

4. Convert the message representative *m* to an encoded message *EM* of length $k - r$ bits where *r* is as defined above:

$$EM = \text{I2BSP}\ (m, k - r)$$

5. If I2BSP outputs "integer too large," then output "decryption error" and stop.

6. Apply the EME-OAEP decoding operation to the encoded message *EM* and the encoding parameters *P* to recover a message *M*:

$$M = \text{EME-OAEP-DECODE}\ (EM, P)$$

7. If the decoding operation outputs "decoding error," then output "decryption error" and stop.

8. Output the message *M*.

Note. It is important that the error messages output in steps 3, 5 and 7 be the same, otherwise an adversary may be able to extract useful information from the type of error message received.

EME-OAEP-DECODE (*EM*, *P*)

Options:	*Hash*	hash function (*hLen* denotes the length in bits of the hash function output)
	MGF	mask generation function
Input:	*EM*	encoded message, a bit string of length at least 2*hLen*+1
	P	encoding parameters, a bit string
Output:	*M*	recovered message, a bit string of length at most $\|EM\|$-2*hLen-1*; or "decoding error"

Steps:

1. If the length of *P* is greater than the input limitation of the hash function (2^{64}-1 bits for SHA-1) then output "decoding error" and stop.

2. If $\|EM\| < 2hLen+1$, then output "decoding error" and stop.

3. Let *maskedSeed* be the first *hLen* bits of *EM* and let *maskedDB* be the remaining $\|EM\|$ - *hLen* bits.

4. Let $seedMask = MGF(maskedDB, hLen)$.

5. Let $seed = maskedSeed \oplus seedMask$.

6. Let $dbMask = MGF(seed, \|EM\| - hLen)$.

7. Let $DB = maskedDB \oplus dbMask$.

8. Let $pHash = Hash(P)$, a bit string of length *hLen*.

9. Separate *DB* into an bit string *pHash'* consisting of the first *hLen* bits of *DB*, a (possibly empty) bit string *PS* consisting of consecutive zero bits following *pHash'*, and a message *M* as

$$DB = pHash' \,\|\, PS \,\|\, 1 \,\|\, M$$

If there is no 1 bit to separate *PS* from *M*, output “decoding error” and stop.

10. If *pHash*' does not equal *pHash*, output “decoding error” and stop.

11. Output *M*.

3.8 Auxiliary Functions

The standard gives functions for converting integers to bit strings and vice-versa as well as the hash and mask generation functions of the preceding section. These are not detailed here.

3.9 Key Transport Mechanism

The example asymmetric encryption scheme is used as follows. The sender uses the encryption transformation of the scheme to encrypt some data. The recipient, after being sent the encrypted data, decrypts the encrypted data using the decryption transformation of the scheme. The asymmetric encryption scheme is used by the key transport schemes.

3.10 Key Agreement Mechanism

The key agreement mechanism is as follows: Each participant uses the encryption transform to encrypt some data using the other party’s public key. Each sends the data to the other. The data is decrypted, then the common key is derived by x’oring the two separate data pieces together.

4. Security Attributes

4.1 Security of Factoring Based Public Key Cryptography

There is an extensive discussion of the security requirements in terms of protecting the private key(s), protection of intermediate values generated during operation, and usage of PKI. In addition, Section 10.4 of the standard discusses the reason for various security requirements during key construction. Sections 10.6 and 10.7 discuss the impact of the choice of e upon security. Section 10.10 gives key size guidelines.

4.2 Security of Key Transport/Agreement Schemes

The issues involving security of key transport and agreement are outlined below. None of these is addressed by the current document, because it specifies an encryption scheme, not a protocol for key transport or agreement. Whether ANSI X9.44 should contain protocols for dealing with these or whether it should refer to ANSI X9.70 is a matter to be decided.

A number of security attributes may be provided by a particular key agreement or key transport scheme. These attributes are:

- Implicit Key Authentication (IKA) – establishes the identity of the other party; knowledge that the other party <u>has the capability</u> of generating the shared key; provided by a static key that is bound to the other party's identity.
- Key Confirmation (KC) – establishes that both parties have actually computed the same shared key; provided by 1) calculating the shared secret value, 2) using the shared secret value to derive a key and 3) using the derived key in a way that can be verified by the other party.
- Explicit Key Authentication (EKA) – knowledge that the other party did indeed calculate the shared key; provided by doing key confirmation in a scheme that provides implicit key authentication.
- Forward Secrecy (FS) – old keys cannot be determined if the new key is compromised; provided by the use of an ephemeral public key.
- Entity Authentication (EA) – assurance about who is the other party in a real-time communication.
- Know-Key Security (K-KS) – assurance that a particular session key will not be compromised as a result of the compromise of other keys.
- Unknown-Keyshare Resilience (U-KS) – a party (A) knows that the other party (B) knows that the key is shared with A, not anyone else.
- Key-Compromise Impersonation (K-CI) – assurance that even if an adversary obtains your private key, the adversary cannot impersonate another party to you.